游易-程序主题分享合辑

知识管理部 等 2022.06.07 15:37 6236 321 184个资源

汇总从101至今的游易征稿文章合辑。

专题 游易程序第125期 手游内存优化

目录

G93内存优化策略

任意飞 2019.09.29 20:17 835 8 3 查看原文

> 为了满足东南亚部分1G内存的移动设备(iphone6/5s)能跑起来本项目，做了一些内存优化措施，希望能抛砖引玉，给大家带来启发。

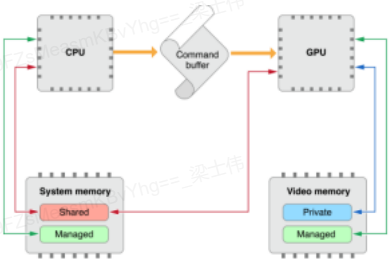
目标:

为了满足东南亚部分1G内存的移动设备(iphone6/5s)能跑起来本项目，做了一些内存优化措施，希望能抛砖引玉，给大家带来启

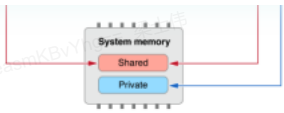
1. iOS内存机制

因为要拯救机型为1G的iphone6/5s, 因此进行具体的优化工作之前, 有必要了解iOS平台上内存的基本机制, 与桌面平台内存机制异, 做到有的放矢.

1.1 内存布局



MacOS



iOS

从上图对比可以明显看出，iOS移动设备是没有物理独立显存结构的。

MacOS:

基于传统桌面平台

采用Discrete Memory Model,

可能具有独立的显存物理介质(区别考虑独立/集成显卡)

具有Shared / Private / Managed三种内存分配模式

iOS:

基于移动设备的SoC

采用特殊的Unified Memory Model

CPU/GPU使用同一块System memory，对于Allocation，可以指定两种模式 -Shared Mode：CPU/GPU均可以访问的地址空间 Mode: GPU专属地址空间

选择策略

Data size	Resource dirtiness	Update frequency	Storage mode
Small	Full	Every frame	`Shared`
Medium	Partial	Every *n* frames	`Managed`
Large	N/A	Once	`Private` (After a blit from a shared source buffer)

Shared: CPU/GPU都具有访问权限。占用同一块内存地址。

Private: 如果是一次性生成后commit到GPU，就不再需要CPU访问的资源（例如普通diffuse贴图），采用Private即可。

Managed: 限于MacOS, 在独立主存和独立显存，各具有一份copy，在需要的时候进行sync。

所以，iOS内存优化要从主存和显存两方面共同入手，保证最终使用总量不超过device limits.

1.2 内存页基础

1.2.1 Page size

OS	Page Size(User space)	Page Size(Physical)
OSX & early version iOS	4KB	4KB
iOS (A7 & A8)	16KB	4KB
iOS (A9 ~ ?)	16KB	16KB

1.2.2 Page Type

Clean page:

Read-Only data (code pages)

File-backed memory

Framework constants

...

Dirty page:

Heap Allocation

App written

...

对于clean page（很多只读的内容，例如code pages）,可以直接被page out，需要的时候再重新从storage上load进来。而对于 则没有这么简单了，例如:

int *array = malloc(20000 * sizeof(int)); array[0] = 32 array[19999] = 64

虽然malloc了多个page，但只有首尾两个被app written的page是dirty的。中间的pages依然是clean的，可以被page out。首尾的 pages，不能被page out，只能被compressed.

active list：近期被access过的pages

inactive list：近期未被access的pages

free list：已释放可被重用的pages

释放的路径:

active list -> inactive list -> free list

Dirty page压缩示例：

->

所以，有时由于Compressor的存在，爆内存的情况趋于复杂化了：释放内存时可能造成内存用量增加。

某种情况需要释放Dictionary的内容，需要先Uncompress它，造成了意外的内存占用。

我们真正关心和需要优化的内存占用:

Memory Footprint = Dirty pages + Compressed pages.

1.3 内存机制总结

iOS：独立显存？ 不存在的。要住就住合租房，内存爆炸一起炸。

iOS: Disk Dirty Page In/Out？不存在的。 不够就挤（Compressed），挤不够就瞅他（Mem Warning），瞅他不动就削他（Terminated）。

2. 内存profile工具

iOS engine: Instruments - Allocation, Leaks

PC engine/scripts, iOS scripts: Magic Sniffer Heap

Py: gc/objgraph/tracemalloc

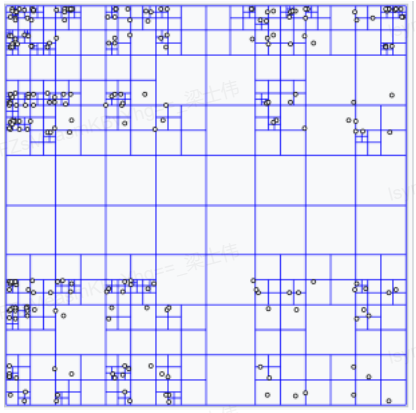
除开脚本层，Instruments的信息是最完整的，勾上Invert Calltree / Hide system Lib / Separated by threads 可以获得比较详细的backtrace的内存追溯分配信息。绝大部分内存问题可以在iOS上主导查到。

3. 优化策略

3.1 Landscape

RenderPatch RenderDetail

由四叉树细分为多层Patch,最近处精细层为Detail层:



ViewPosition + ViewRange控制加载范围

1 * Patch = 128 * 128 Grid

定长步长采样形成多层patch, 具体采用哪层patch

然而,层N的patch,是否符合加载精度要求, 取决于landscape.screen_space_error_bound容错阈值.

初始化一个预设的optimal的chunk块

比较screen_space_error_bound:

* 若不满足,添加IncreaseLodTask异步任务,加载四叉树中更加精细的N+1层块

* 若超过,添加DecreaseLodTask,加载四叉树中更粗糙的N-1层块,直到获取到对应层.

该阈值的对比标准,取决于chunk块的bounding_box屏幕空间占比

之前该值=150左右,导致全场景几乎所有的chunk加载的detail层,浪费大量内存.

根据大场景实际情况,取合理的landscape.screen_space_error_bound(我们的场景约4km * 4km, 取值1500).得到合适的加载patc

3.2 Dynamic Preload

大世界场景，具有较多的"预加载模型":

* 远景高楼 (Y轴较大)

* 大型桥梁 (XZ轴占面积较广)

* 大型地表

1 model in 1 chunk，UUID conflict。

1 model attach to 1 node

> 方案B：每一定逻辑帧间隔，循环遍历检测场景中所有preload物体与角色可视域相交测试。

决定是否Load / Unload 相关preload model

全场景150~200总量的preload models，轮询可以接受。

采用以下参数控制动态预加载:

> preload_check_tick: 加载间隔

preload_extend_dist: 外扩自定义距离

preload_y_limit: Y轴最低加载阈值,高于该阈值的model无视bounding_box直接加载

*轮询准则

> XZ域：

设M包围盒为(Bx,By,Bz)

自定义外扩距离 Dcustom

PosP in Rectangle(Bx + Dcustom, Bz + Dcustom)

PosP.y < Ymax

二者满足其一, 便启动异步加载线程加载model

3.3 Vertex streams discard

> G93特点: 复杂的场景，较多的场景物件，复杂的机甲模型 - 大量的顶点数据

* 低内存模式下，抛弃部分顶点数据

对于<=1G内存的低端机，美术效果已经在低配中大打折扣。

Vertex stream data: pos / normal / tangent / color, etc.

对于低配机，加载时动态抛弃 normal, tangent, color数据。

大部分材质的低配版，不使用normal/tangent/color。对于小部分要使用的，给缺省值代替，效果有一定artifact能接受。

脚本接口，动态根据不同材质，设置是否抛弃部分顶点数据。

* 尝试动态load/unload MeshVertexData:

现象: 角色环顾一周，内存增加

引用计数的MVD，Mesh第一次进入时AllocBloc分配主存空间

动态load/unload后卡顿情况增加较多

遂放弃。

* keep_lod_meshvertexdata：

为减少卡顿进行的load数据保留加载。低内存模式下关闭，lod加载触发时，才加载进内存。

3.4 Shader compiled data

正常流程：

加载：gl::CreateShader->gl::ShaderSource->gl::CompileShader->gl::LinkProgram;

释放：gl::DetachShader -> gl::DeleteShader

老NeoX引擎由于部分Tegra Problems，Link成功后，并没有detach + delete compiled data。较大内存浪费（视材质 宏组定）

动态开关，释放compiled data

3.5 Gis async loading

角色动作数据量较大（主角400+），单个gis数据较大。

启用引擎的gis拆分+异步加载。

bnk中尽量使用stream的流式音效, 对于高配音效,采用stream后常驻cache的方式.

没有特殊原因,音效一般不直接load常驻内存.

3.7 Skip Mipmaps

对于低内存模式的机器，贴图内存占用，通过动态的加载跳过前几级mipmap，释放显存容量占用。设计了三种模式：

render.set_texture_skip_level：全局贴图skip等级

render.set_texture_skip_level_by_tag：根据自定义标记（场景，人物，机甲）的区分，进行不同的skip等级。

render.set_texture_skip_level_by_rank：根据贴图的信噪比PSNR进行选择阈值skip等级。

合理设置skip等级，能大量降低显存占用，在移动设备的unified memory model的框架下，等同于减少整体内存峰值。

3.8 Sfx level + Culling

* world.set_sfx_render_level: 设置合适的特效等级，减少多余特效层的加载

* world.set_sprites_percent: 设置合适的特效粒子产生百分比，减少粒子发射总数量

* 对于特效使用远景裁剪面优化，减少可视域以外特效的渲染内存占用开销

3.9 Font cut

仅保留常用最基本的字体文件(7M)

区分的粗体字体在低内存模式下不加载，以常规字体替代

3.10 Python threads pool

在iOS 12+的Instruments里,可以看到,每启动一个python线程,主存会分配一个4MB的stack for Python VM.

该stack大部分是占用clean的page，小部分dirty page是真实占用，并不会因为N *4MB导致内存memory warning

增加一个简单的threadpool，复用python thread，减少总的stack分配峰值。

3.11 GC check

结合objgraph / gc / tracemalloc，进行各项目逻辑层相关的常规py内存检查，最好固定几个常用use case，QA定期跑查后输出g
提前获知循环引用和其他错误allocation的地方。

4. References

<<iOS Memory Deep Dive>> - developer.apple.com

<<Metal Best Practices Guide>> - developer.apple.com

最热 最新

Const(傅鹏)

3楼 牛!

2020-04-21 11:49 回

正元(史正元)

2楼 

2020-03-09 11:54 回

匿名

1楼 

2020-03-05 16:03 回

加载完毕,没有更多了